Dual Optimization of Conditional Probability Models

December 21, 2006

Roland Memisevic
Department of Computer Science,
University of Toronto,
roland@cs.toronto.edu

Abstract

We describe a dual formulation for conditional probability models operating in joint features spaces. In the dual we derive a very fast training scheme akin to John Platt's "sequential minimal optimization", an efficient standard method for training support vector machines. The algorithm is closely related to perceptron learning and extends the applicability of probabilistic discriminative models to online settings and to large scale learning problems.

1 Introduction

Discriminative methods are among the most successful approaches in machine learning, because they provide a means of constructing systems entirely 'by example'. In the last few years especially kernel based models have gained a lot of popularity, because of their ability to construct non-linear feature spaces at a cost that does not scale with dimensionality.

Much work on kernel based models has originally focused on somewhat restricted problems, such as classification with binary labels, or regression with one-dimensional outputs. However, recent work has extended the applicability of discriminative methods to far more general problems, for example from binary- to multiclass classification (*e.g.* [4]), and, more recently, beyond single outputs, to the prediction of structured objects (*e.g.* [9], [12]), for which previously mainly generative models have been used.

A simple common perspective, that unifies most of the extensions to these more general settings, can be derived from the perspective of *joint feature spaces*. According to this view, an input-output-pair $(\boldsymbol{x}, \boldsymbol{y})$ is mapped to a joint feature representation $\phi(\boldsymbol{x}, \boldsymbol{y})$ and a linear *score* $\boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}, \boldsymbol{y})$ is used to measure the compatibility of $\boldsymbol{x}$ and $\boldsymbol{y}$. Inference consists in, given $\boldsymbol{x}$, finding the output $\boldsymbol{y}$ that maximizes the score. Learning amounts simply to adapting $\boldsymbol{w}$ based on a training set $\{(\boldsymbol{x}^{\boldsymbol{i}}, \boldsymbol{y}^{\boldsymbol{i}})\}_{i=1,\dots,N}$.

Most linear discriminative models can be classified as being either 'margin'- based or probabilistic, depending on the loss-function they use (hinge-loss or negative log probability, respectively). Examples of margin-based methods are standard support vector machines (svm) and recent extensions, such as multiclass svms [4], maximum-margin-Markov networks [12], and other general joint-feature based models (*e.g.* [13]). Probabilistic models are usually based on extensions of logistic regression and include, for example, multinomial logistic regression and conditional random fields [9]. Even though both classes of method usually perform similarly in terms of error rates[1], they do show some practical differences. Margin based methods are naturally well suited to optimization in the *dual*, because training for them is most naturally cast as a constrained optimization. The dual formulation in turn allows for the use of kernels, because data-points enter it only in terms of

[1]Strictly speaking, probabilistic models maximize a margin, too. They, too, attempt to increase confidence in the correct answer while decreasing confidence in wrong answers.

inner products. More importantly, the dual formulation makes available a fast and convenient optimization procedure, John Platt's well-known "sequential minimal optimization" (smo). Since the algorithm is very fast and relatively easy to implement, it is currently one of standard methods for training margin-based methods.

Probabilistic models provide somewhat orthogonal advantages. Since their predictions come with 'error-bars', they lend themselves to easier interpretation and can be combined into complex larger-scale models. While kernels can also be introduced into general probabilistic models, they usually rely on (generalized versions of) the representer theorem, which states, that the optimal parameters can be written as a kernel expansion (see *e.g.* [15], [10]). Probabilistic models are therefore often trained by gradient-based *primal* optimization of that dual representation.

In this paper we derive a dual formulation for general probabilistic kernel regression models. We consider the general case of joint feature spaces, and derive duals for multiclass learning, as well as more recent models, such as conditional random fields, and potential other extensions. In related work, [7] have pointed out the close relation between margin-based and probabilistic classification methods using conjugate duality. Here, we focus is explicitly on the involved constrained optimization problem, with the main goal to clarify the close relation of probabilistic models to a parallel line of recent work on margin based methods (*e.g.* [12], [13]).

In this paper we restrict our attention to methods without 'bias'-term, that is methods in which the score is a pure linear (not affine) function of the features[2]. The main advantage is that it makes our approach similar to the above mentioned parallel line of work on margin-based methods. Furthermore, our method can also be interpreted as a (probabilistic) version of the Adatron-algorithm [6] which in turn can be thought of as a dual kind of perceptron learning. The method can be run in an online-processing mode, in which updates are performed on a point-by-point basis. This reduces the memory footprint from quadratic to linear in the number of data-points and therefore makes it possible to apply the model to much larger datasets than is possible, when using the standard, primal, optimization.

[2]While we could easily accommodate a bias indirectly by extending the feature vector (or correspondingly a kernel), we would argue that in many cases (when using Gaussian kernels *e.g.*) it is not actually necessary to do so.

In a related setting, [8] have recently considered an smo-like optimization procedure for *binary* logistic regression and demonstrated significant gains in efficiency over conventional optimization. Our method can be viewed as a generalization of that work to arbitrary joint feature settings, including in particular multiclass learning. Alternative extensions to multiclass settings have been pursued previously by [14] and [5] (however the latter approach turned out to be too slow to be useful in practice).

2 Logistic regression

Probabilistic classifiers model the conditional distribution $p(y|\boldsymbol{x};\boldsymbol{w})$ over labels y, given an input feature vector $\boldsymbol{x}$. (We restrict our attention to scalar class labels here and denote them by plain font letters. We use bold font to denote vectors.) At test time, classification decisions can – if desired – be made on test cases $\boldsymbol{x}^{\text{test}}$ by maximizing this probability wrt. y, that is by setting $y^{\text{test}} = \arg\max_y p(y|\boldsymbol{x}^{\text{test}};\boldsymbol{w})$. The distribution p is commonly modeled using a linear *score* $\boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x},y)$ plugged into the 'softmax' response function:

$$p(y|\boldsymbol{x};\boldsymbol{w}) = \frac{\exp(\boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x};y))}{\sum_y \exp(\boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x};y))}. \tag{1}$$

Here, $\phi(\boldsymbol{x},y)$ is a joint feature vector for the input-output pair $(\boldsymbol{x},y)$, and y takes on values in a set of class labels $\{1,\dots,C\}$.

The standard way of fitting this model based on training data $\{(\boldsymbol{x}^i,y^i)\}_{i=1,\dots,N}$ is by minimizing the 'logistic loss', defined as the negative log-probability $l^{\log}(\boldsymbol{x}^i,y^i,\boldsymbol{w}) = -\log p(y^i|\boldsymbol{x}^i;\boldsymbol{w})$, averaged over the training set. Plugging in (1) we get the per-case loss

$$l^{\log}(\boldsymbol{x}^i,y^i,\boldsymbol{w}) = \log\sum_y \exp(\boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}^i,y) - \boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}^i,y^i)). \tag{2}$$

A common way of selecting a classifier that can generalize well is to additionally assume a Gaussian prior on $\boldsymbol{w}$, and then to maximize the corresponding MAP estimate for the parameters, which simply amounts to minimizing the *regularized loss*:

$$J(\boldsymbol{w}) = \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \sum_i l^{\log}(\boldsymbol{x}^i, y^i, \boldsymbol{w}). \qquad (3)$$

Usually the regularized loss is optimized directly wrt. $\boldsymbol{w}$, using *e.g.* gradient based optimization. Here we deviate from the standard approach, and instead of minimizing Eq. 3, we consider the equivalent convex constrained optimization problem:

$$\min_{\boldsymbol{w},\xi} \quad \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \sum_i \log \sum_y \exp(\xi_y^i)$$
$$\text{s.t.} \quad \xi_y^i = \boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}^i; y) - \boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}^i; y^i) \quad \forall i, y \; (4)$$

(Note that the constraints ensure trivially that solving 4 is equivalent to minimizing 3.)

While replacing an unconstrained problem by an equivalent constrained one might seem wasteful at first glance, it allows us to obtain a perspective onto the model that parallels the view taken in margin-based methods. In particular, we can now consider the dual of Problem 4 (see Appendix 6), which takes the simple form:

$$\max_{\alpha} \quad -\frac{1}{2\lambda}\alpha^T K \alpha + \frac{1}{\lambda}\alpha^T K \delta + \sum_i H(\alpha^i) \qquad (5)$$

$$\text{s.t.} \quad \alpha_y^i \geq 0 \quad \forall i, y; \quad \sum_y \alpha_y^i = 1 \quad \forall i, \qquad (6)$$

where $H(\cdot)$ is the Shannon entropy, $\delta = (\delta_{y,y^i})_{(i,y)}$ and K is the matrix of inner products $\phi(\boldsymbol{x}, y)^{\mathrm{T}}\phi(\boldsymbol{x}, y')$ between the joint feature vectors.

The dual is a convex problem and it takes a form similar to the dual optimization problem for training support vector machines. One difference is the entropy-term, which gives it a general nonlinear form, while the svm-dual is a quadratic program. This nonlinearity has technical consequences if we want to adopt svm-optimization procedures, as we discuss in Section 3.

One advantage of considering the dual as opposed to the primal problem is that the feature vectors $\phi(\boldsymbol{x}^i, y)$ appear only in the form of inner products (summarized in the Grammian K). We can therefore replace these by kernel function evaluations, and fit the model in the corresponding feature space. Technically, we simply replace the inner product matrix K by a kernel matrix, derived from a (*joint*) kernel $k((\boldsymbol{x}, y), (\boldsymbol{x}', y'))$. The linear score then takes the form (using Eq. 24):

$$\boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}^i, y) = -\frac{1}{\lambda}\sum_{j,y'}(\alpha_y^j - \delta_{y',y^j})k((\boldsymbol{x}^i, y), (\boldsymbol{x}^j, y')). \qquad (7)$$

Kernel versions of probability models have been considered previously *e.g.* in [15] and [1]. In contrast to our approach, those models rely on the representer theorem, that ensures that the minimizer of Eq. 3 takes the form of a kernel expansion. As a consequence, usually gradient based optimization in the primal (using the dual representation) is used to train the models.

3 Sequential minimal optimization

The similarity of the constrained problem to that of margin based methods demonstrates that these methods are actually closely related. A practical implication is that we can apply similar learning procedures to these models. In the following we show how we can derive an optimization scheme for conditional probability models that is similar to John Platt's "sequential minimal optimization" (smo) [11], a fast and simple training method for support vector machines, and the standard method used in most svm software-packages.

We consider a version of the smo algorithm similar to the method suggested by [12] for solving structured max-margin problems. Following the argumentation there, we suggest cycling through the training data and iteratively solving Problem 5 for only a single α^l one at a time, effectively performing coordinate ascent in the objective function. This is possible, since the constraints couple only the variables α_y^i for each i, *ie.* no constraint affects any α_y^i, $\alpha_{y'}^j$ for $i \neq$

j. Assuming that we start with a feasible α^l (which can be achieved trivially by setting for example $\alpha_y^i = (1/C)\ \forall i, y$) all we need to make sure then is that the update for α^l does not destroy feasibility.

We can achieve such an update as follows: We set (for each y) $\alpha_y^l \leftarrow \alpha_y^l + \nu_y$, where we constrain the additive update ν_y, such that $(\sum_y \nu_y = 0)$ and $(\alpha_y^l + \nu_y \geq 0, \forall y)$ hold. Then, adding ν_y will keep the sum over all α^l unchanged and will maintain positivity. Performing the update so as to maximize the objective leads to the following problem in the variables $\nu := (\nu_y)_y$ (we abbreviate $K_{yy'}^{ij} = k((\boldsymbol{x^i}, y)(\boldsymbol{x^j}, y'))$ in the following):

$$\begin{aligned} \max_\nu \quad & -\frac{1}{\lambda}\sum_{y,y'} \nu_y \nu_{y'} K_{yy'}^{ll} \\ & +\frac{1}{\lambda}\sum_y \nu_y B_l - \sum_y (\alpha_y^l + \nu_y)\log(\alpha_y^l + \nu_y) \\ \text{s.t.} \quad & \sum_y \nu_y^l = 0; \quad \nu_y^l \geq -\alpha_y^l \quad \forall y \end{aligned} \tag{8}$$

where B_l is a constant:

$$B_l = \sum_j K_{y,y^j}^{l,j} - \sum_{j \neq l, y'} \alpha_{y'}^j K_{y,y'}^{l,j} - 2\sum_{y'} \alpha_{y'}^l K_{y,y'}^{l,l}$$

While until now, we have not gained a direct advantage from restricting the problem to coordinate-wise optimization, we can now exploit the fact that additional structure is present in the problem: All α_y^l are constrained to form a *distribution* (or correspondingly, the ν_y sum to zero). The core insight of smo-like algorithms is, that it is natural to solve problems under these constraints wrt. to only *pairs* ν_{y^1}, ν_{y^2} one at a time. The reason is that, to keep the constraint $\sum_y \nu_y = 0$ satisfied, we just need to make sure, that $\nu_{y^1} = -\nu_{y^2}$ holds true. In other words, we just need to *move weight* from one variable to another, which ensures automatically that the net weight stays the same. That means, however, that the update can be achieved using a *single* variable: We can define the 'weight-change' $\Delta = \nu_{y^1} = -\nu_{y^2}$ and solve for Δ. Performing this substitution in Problem 8 yields a simple one-dimensional problem of the form:

$$\begin{aligned} \max_\Delta \quad & -a\Delta^2 - b\Delta \quad - \quad (\alpha_{y^1}^l + \Delta)\log(\alpha_{y^1}^l + \Delta) \\ & - \quad (\alpha_{y^2}^l - \Delta)\log(\alpha_{y^2}^l - \Delta) \\ \text{s.t.} \quad & -\alpha_{y^1}^l \leq \Delta \leq \alpha_{y^2}^l \end{aligned} \tag{9}$$

with constants

$$\begin{aligned} a &:= \frac{1}{2\lambda}(K_{y^1y^1}^{ll} + K_{y^2y^2}^{ll} - 2K_{y^1y^2}^{ll}), \\ b &:= \frac{1}{\lambda}(c(y^2) - c(y^1)), \\ c(y) &:= \sum_j K_{yy^j}^{lj} - \sum_{j,y'} \alpha_{y'}^j K_{yy'}^{lj}(1 + \delta_{jl}). \end{aligned}$$

We obtain a simple convex problem with box-constraints. The presence of the logarithm seems to imply a degeneracy when the box-constraints are satisfied with equality. However, it is easy to show that the optimum always lies the interior of the constraint region, which follows, for example, from the fact that all constraints in the primal (Problem 4) are equality constraints. A similar observation was made by [8] for binary logistic regression.

It is interesting to compare Problem 9 to the one-dimensional sub-problem in the corresponding svm-formulation of smo. The fact that the svm-dual is a quadratic program leads to a corresponding one-dimensional sub-problem that can be solved analytically (followed by clipping, because of constraints similar as here [12]; [11]). Here, we obtain a nonlinear problem, that we need to solve iteratively instead. The natural way of solving the problem is therefore by using Newton-Raphson iterations. A legitimate question is, whether a significant speed-up over more traditional optimization procedures is also achievable here, despite the more demanding iterative optimization. We show in section 5 that this is clearly the case.

Three decisions that need to be made to complete the algorithm are (1) in what order to pick data-points to solve for, (2) how two choose pairs (y_1, y_2) within each coordinate ascent step (3) when to stop. Natural solutions for problems (2) and (3) follow directly from the KKT-conditions. We discuss these below in Subsection 3.1. Regarding problem (1), many choices are possible. The most simple one is simply *cycling* through a randomly ordered dataset and performing a fixed number D of optimizations for each data-point. We have chosen this strategy for our experiments, and obtained very good results. However, more sophisticated strategies for point-selection might help improve the procedure further and point to possible future research.

An important advantage of the method is that the memory-requirement is reduced from quadratic in the number of data-points for traditional, gradient based optimization, to basically linear in the number of data-points. The reason is that running smo needs typically only a couple of sweeps over the dataset in order to converge. More importantly, in each single iteration only a *single row* of the kernel matrix is needed (the l^{th} row – see Problem 9). Gradient based optimization, on the other hand, requires the computation of essentially *every entry* of the kernel matrix for every single function- and gradient-evaluation. The only way to rein in the time-complexity is therefore a pre-computation of the kernel-matrix. The small memory-footprint is therefore a standard motivation for using smo over other optimization methods. The time-complexity is generally quadratic for both methods, but with a much smaller constant for smo, which is equal to the number of sweeps over the data-set that are required for convergence.

3.1 Pair selection and stopping criteria

At the solution we need to satisfy the KKT-conditions (Eqs. 25 and 26, Appendix 6). (Note that we have used Eq. 24 in the representation of $\boldsymbol{w}$). We can combine the conditions in Eqs. 25 and 26 to get:

$$\log \alpha^i_y - \boldsymbol{w}^{\mathrm{T}}\phi(x^i, y) = -\log \sum_{y'} \exp(\boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x^i}, y')) \quad \forall y \tag{10}$$

A crucial insight now is that the RHS does not depend on y. Therefore, all we need to make sure is that the LHS is *the same* for all y:

$$g^l(y) := \log \alpha^l_y - \boldsymbol{w}^{\mathrm{T}}\phi(x^l, y) = c^l \quad \forall l, y \tag{11}$$

for some (arbitrary) constant c^l. As a result, we obtain criteria for both, pair selection and stopping: We iteratively set $y_1 = \arg\min_y g^l(y)$ and $y_2 = \arg\max_y g^l(y)$ and solve the restricted dual for the pair y_1, y_2. Optimality is achieved when $g^l(y)$ is constant across y. But using y_1 and y_2 from above, this means that we can stop as soon as $g^l(y_2) = g^l(y_1)$ (within numerical tolerance). We therefore define the 'optimality gap'

$$G := g^l(y_2) - g^l(y_1) \tag{12}$$

and stop as soon as G gets smaller than some predefined tolerance ϵ. (In our experiments we have set $\epsilon = 10^{-6}$.)

Algorithm 1 smo for probability models

while $G > \epsilon$ **do**
  Pick next data-point l (according to schedule).
  for $i = 1 : D$ **do**
    $y_1 = \arg\max_y g^l(y)$
    $y_2 = \arg\min_y g^l(y)$
    Set constants a and b for 1-d problem (Eq. 9)
    Solve 1-d problem
  end for
  Set $G = \max_y g^l(y) - \min_y g^l(y)$
end while

It is important to note that the pair selection and stopping criteria involve only a simple minimization/maximization. This observation provides essentially the grounds for potential extensions to structured learning settings, in which these procedures can be performed using *e.g.* a form of belief propagation.

Note that we can express $g^l(y)$ in terms of only α as (using Eq. 7):

$$g^l(y) = \log(\alpha^l_y) + \frac{1}{\lambda}\sum_{j,y'}(\alpha^j_{y'} - \delta_{y'y^j})k((\boldsymbol{x}^l, y)(\boldsymbol{x}^j, y')). \tag{13}$$

The overall optimization procedure is summarized in Algorithm 3.1.

4 Joint kernels

Until now, we have defined the model and optimization in terms of general joint kernel functions

$k((\boldsymbol{x}, y), (\boldsymbol{x'}, y'))$. Recent work (see *e.g.* [13]) has shown how a variety of different learning tasks can be captured in the framework of joint feature learning. In the following (Subsection 4.1) we discuss the special case of *multiclass* learning in detail, and describe how to apply smo in this setting. In Subsection 4.2 we show briefly, how *multilabel* classification can be treated as another special case within the framework.

4.1 Generic multiclass kernels

Multiclass learning is a special case of learning in joint feature spaces that we obtain by using joint kernels of the form:

$$k((\boldsymbol{x^i}, y), (\boldsymbol{x^j}, y')) = \delta_{y,y'} k^{\boldsymbol{x}}(\boldsymbol{x^i}, \boldsymbol{x^j}), \qquad (14)$$

where $k^{\boldsymbol{x}}(\boldsymbol{x^i}, \boldsymbol{x^j})$ is any kernel defined in the input space. For example, we could define $k^{\boldsymbol{x}}(\boldsymbol{x^i}, \boldsymbol{x^j})$ as an RBF-kernel: $k^{\boldsymbol{x}}(\boldsymbol{x^i}, \boldsymbol{x^j}) = \exp(-\frac{1}{\sigma^2}\|\boldsymbol{x^i} - \boldsymbol{x^j}\|^2)$, or another standard kernel, such as polynomial. It is easy to show how these kernels recover standard formulations of multiclas problems by plugging (14) into (7).

Learning with smo using kernels of this form is straightforward. We can directly apply Algorithm 3.1, using Eq. 14, when computing the quantities $g^l(y)$, a, and b. Note in particular, that the score (Eq. 7) under this choice of kernel simplifies to

$$\boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}, y) = \frac{1}{\lambda}\Big(\sum_{j:y=y^j} k(\boldsymbol{x}, \boldsymbol{x^j}) - \sum_j \alpha_y^j k^{\boldsymbol{x}}(\boldsymbol{x}, \boldsymbol{x^j})\Big). \qquad (15)$$

Furthermore, Eq. 13 simplifies to

$$g^l(y) = \log(\alpha_y^l) + \frac{1}{\lambda}\Big(\sum_j \alpha_y^j k^{\boldsymbol{x}}(\boldsymbol{x^l}, \boldsymbol{x^j}) - \sum_{j:y=y^j} k(\boldsymbol{x^l}, \boldsymbol{x^j})\Big), \qquad (16)$$

and the constants in Problem 9 to

$$\begin{aligned} a &= \frac{1}{\lambda} k^{\boldsymbol{x}}(\boldsymbol{x^l}, \boldsymbol{x^l}) \\ c(y) &= \sum_{j:y^j=y} k^{\boldsymbol{x}}(\boldsymbol{x^l}, \boldsymbol{x^j}) - \sum_j \alpha_y^j k^{\boldsymbol{x}}(\boldsymbol{x^l}, \boldsymbol{x^j})(1 + \delta_{jl}). \end{aligned}$$

4.2 Structured kernels

A useful extension to basic classification consists in using *label vectors* $\boldsymbol{y}$ as opposed to scalar labels y. Conditional random fields (crf) are linear probabilistic models for this purpose [9]. Predicting label vectors entails a combinatorial explosion, that makes the problem intractable in general, and crfs deal with this problem by restricting attention to certain decomposing feature vectors that allow for the use dynamic programming for learning and prediction. [10] and [1] use a generalized version of the representer theorem to show how crfs can be defined in kernel feature spaces.

In the following we show how we can derive kernel crfs alternatively using the dual optimization perspective. Intractabilities during learning arise in the dual Problem 5, because of the presence of a number of dual parameters $\alpha_{\boldsymbol{y}}^i$ that is exponential in the number of labels (*i.e.* the dimensionality of $\boldsymbol{y}$). A simple trick that makes the problem tractable is to arrange the entries of $\boldsymbol{y}$ in a graph, and to let the kernel *decompose* as a sum of 'sub'-kernels defined only on the cliques of the graph:

$$k((\mathbf{x}, \mathbf{y}), (\mathbf{x}', \mathbf{y}')) = \sum_{m,n} k((\mathbf{x}, y_m), (\mathbf{x}', y_n')), \qquad (17)$$

where m and n range over the cliques of the graph. This decomposition then allows us to use a trick similar to the one used in [12], and to derive a tractable representation for crfs. Using Eq. 17 we can re-write the dual representation of the scores formally as

$$\begin{aligned} & \boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x^i}, \boldsymbol{y}) \\ = \quad & -\frac{1}{\lambda}\sum_{j,\boldsymbol{y'}} \alpha_{\boldsymbol{y'}}^j k((\boldsymbol{x^i}, \boldsymbol{y}), (\boldsymbol{x^j}, \boldsymbol{y'})) \\ = \quad & -\frac{1}{\lambda}\sum_j \sum_{m,n} \sum_{\boldsymbol{y'}} \alpha_{\boldsymbol{y'}}^j k((\boldsymbol{x^i}, y_m), (\boldsymbol{x^j}, y_n')) \\ = \quad & -\frac{1}{\lambda}\sum_j \sum_{m,n} \sum_{y_n'} k((\boldsymbol{x^i}, y_m), (\boldsymbol{x^j}, y_n')) \sum_{\sim y_n'} \alpha_{\boldsymbol{y'}}^j \\ =: \quad & -\frac{1}{\lambda}\sum_j \sum_{m,n} \sum_{y_n'} k((\boldsymbol{x^i}, y_m), (\boldsymbol{x^j}, y_n')) \mu_{y_n'}^j, \end{aligned}$$

where we introduce a (polynomially sized) set of *marginal variables* $\mu_{y_n'}^j$, that allow us to express the problem in tractable form (we use the 'notsum'-notation ($\sim y_n$) to denote marginalization). Similarly, we also get tractable representations of the quadratic and linear terms of the dual objective (Eq. 5). We also need to make sure, however, that the marginal variables are consistent, that is, that they agree on the nodes that they share. We will show in the following, how we can achieve this implicitly.

Note, that due to the density constraints on the α^l (Eq. 6), the marginal variables are simply *marginal distributions* on the cliques of the underlying graph, *ie.* we obtain a probabilistic graphical model in the dual. Consistency can be achieved easily only for triangulated graphs[3], and we restrict our attention to these in the following. In other words, we consider only marginals $\mu_{r,s}$ and μ_r defined on the edges (r, s) and nodes r of a tree, respectively. Now, using the junction tree theorem [3], for each l we can express in general:

$$\alpha^l_{\mathbf{y}} = \frac{\prod_{(r,s)\in E} \mu^l_{y_r,y_s}}{\prod_{r\in V} \mu^l_{y_r}}. \tag{18}$$

Under this representation the entropy term in the dual decouples as $-\sum_i \left[\sum_{r\in V} H(\mu^i_{y_r}) + \sum_{(r,s)\in E} I(\mu^i_{y_r,y_s})\right]$, where $I(\cdot)$ is the mutual information. Note that the dual objective could be computed tractably as a result.

In practice it is not actually necessary to be able to do so, however, since to run smo, all we ever need to compute is the 'argmax' of $g^l(\boldsymbol{y})$, and the constants a, b in Problem 9. Using the marginal representation, we can use belief propagation for this purpose. (Note in particular, that the 'log'-term for computing $g^l(\boldsymbol{y})$ in Eq. 13 decouples similarly as the entropy above). In other words, Algorithm 3.1 carries over with essentially no change to the structured learning setting.

After having solved the one-dimensional problem, we can perform the update in terms of the marginals by adding Δ, or $-\Delta$, to each marginal that is compatible with $\boldsymbol{y^1}$, or $\boldsymbol{y^2}$, respectively. Formally:

$$\begin{aligned}\mu^l_{y_r,y_s} &= \mu^l_{y_r,y_s} + \delta_{y_r y^1_r}\delta_{y_s y^1_s}\Delta - \delta_{y_r y^2_r}\delta_{y_s y^2_s}\Delta,\\ \mu^l_{y_r} &= \mu^l_{y_r} + \delta_{y_r y^1_r}\Delta - \delta_{y_r y^2_r}\Delta.\end{aligned}$$

[3]For graphs that are not triangulated we could proceed and solve the dual *approximately*, opening up a huge field of potential variational learning approaches.

5 Experiments

We discuss some simple experiments to assess the efficiency of smo training for multiclass problems. The datasets that we have used (taken from the UCI-database and USPS-digits) are summarized in Table 1 (N^{test} and dim refer to the number of test points and the input space dimensionality, respectively). All datasets have been normalized to unit-variance in each dimension. We have used RBF-kernels with bandwidths as described below. We have compared against the standard approach of solving the *unconstrained* primal problem obtained from the representer theorem. To solve the primal we have used conjugate gradients. In particular, we used Carl Rasmussen's function 'minimize' as the optimizer[4]. Conjugate gradients is known to be one of the most efficient methods available for training logistic regression and 'minimize' probably one of the 'tougher' competitors to compare against. All experiments were run on a four-processor Pentium4 machine with four Gigahertz processors and four Gigabyte RAM.

Figure 1 shows learning curves on the USPS-digits for three different values of λ (where we have fixed $\sigma^2 = 128.0$). The plots show that overall smo converges much faster on this dataset than conjugate gradients. Note that the larger the value of λ, the faster the initial optimization for conjugate gradients (*ie.* the steeper the initial descent). A similar observation holds true for smo. At the same time, however, continuing optimization is much slower when λ is large for conjugate gradients, while smo converges to the final solution almost immediately (which amounts to performing a few sweeps over the dataset) in all cases. In a second experiment, we have compared absolute training times on several datasets, using the following comparison scheme: We have run conjugate gradients until the change in the primal objective was smaller than 10^{-8}. We have then run smo (using the same random initialization that we have used for conjugate gradients), stopping after the objective[5] had

[4]*All* implementations are in matlab.

[5]Note that computing the primal objective is not actually necessary for running smo. We have computed the primal objective after each sweep over the whole dataset for smo, getting a slightly unfair comparison in favor of conjugate gradients.

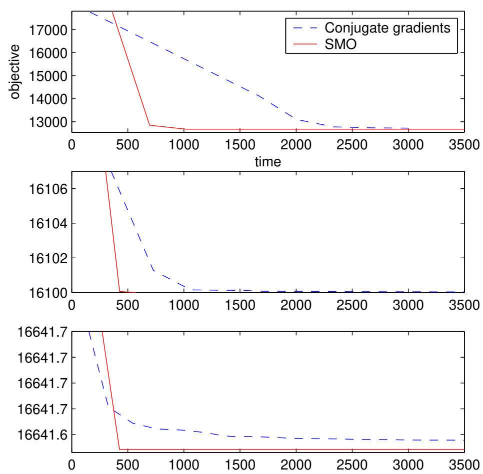

Figure 1: Learning curves. $\frac{\lambda}{N} = \{0.01, 0.1, 1.0\}$ (top to bottom). Cpu-times are shown in seconds.

reached a value *below* the final objective that we obtained from conjugate gradients. The resulting cpu-times (in seconds), along with optimal bandwidths and the best achieved test errors, are recorded in Table 2 and demonstrate a huge gain in efficiency when using smo.

To test the possibility of performing online processing, we have also used smo to train a model on the letter dataset, containing 15000 training examples. We have trained online, *ie.* without storing the kernel matrix. (Note that conventional training would have been difficult for this dataset.) We used $\sigma^2 = 0.1$ and $\lambda = 0.01$. Training until the smo-stopping criterion ($G < 10^{-6}$) was satisfied took a little over 2 days, with a resulting final error rate of 4.7%.

6 Discussion

We have considered probabilistic classification from the perspective of constrained dual optimization, revealing several analogies to margin-based classification. As an example application, we have shown how we can adapt the smo-algorithm to the probabilistic setting. There are many directions for potential future work. Especially interesting ones are extensions to continuous valued settings and the use of variational approximations in the dual. A more technical issue is the investigation of alternative scheduling approaches for smo that could potentially lead to an even further increase in efficiency.

While we have considered only conditional models in this paper, all the derivations could be applied to non-conditional, generative models, by essentially dropping the '$\boldsymbol{x}$' everywhere.

Table 1: Summary of datasets.

Dataset	N	N^{test}	dim	C
WINE	120	58	13	3
GLASS	130	84	9	7
VEHICLE	600	246	18	4
SEGMENT	1500	810	19	7
DIGITS	5000	2000	256	10
LETTER	15000	5000	16	26

A Derivation of the dual

We define $f^i(\boldsymbol{\xi^i}) = \log \sum_y \exp(\xi^i_y)$ and $\boldsymbol{\psi^i} = \left(\boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x^i}, y)\right)_y$ and consider the Lagrangian:

$$L(\boldsymbol{w}, \boldsymbol{\xi}, \boldsymbol{\alpha}) = \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \sum_i f^i(\boldsymbol{\xi^i}) + \boldsymbol{\alpha^i}^{\mathrm{T}}(\boldsymbol{\psi^i} - \mathbf{1}\boldsymbol{\psi}^{\boldsymbol{i}}_{y^i} - \boldsymbol{\xi^i}) \tag{19}$$

Convexity and strict feasibility now allow us to write

$$\begin{aligned}
& \min_{\boldsymbol{w},\boldsymbol{\xi}} \max_{\boldsymbol{\alpha}} L(\boldsymbol{w}, \boldsymbol{\xi}, \boldsymbol{\alpha}) && (20)\\
= \; & \max_{\boldsymbol{\alpha}} \min_{\boldsymbol{w},\boldsymbol{\xi}} L(\boldsymbol{w}, \boldsymbol{\xi}, \boldsymbol{\alpha}) && (21)\\
= \; & \max_{\boldsymbol{\alpha}} \min_{\boldsymbol{w}} \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \sum_i \boldsymbol{\alpha^i}^{\mathrm{T}}(\boldsymbol{\psi^i} - \mathbf{1}\boldsymbol{\psi}^{\boldsymbol{i}}_{y^i}) \\
& - \sum_i \max_{\boldsymbol{\xi^i}} \boldsymbol{\xi^i}^{\mathrm{T}}\boldsymbol{\alpha^i} - f^i(\boldsymbol{\xi^i}) && (22)\\
= \; & \max_{\boldsymbol{\alpha}} \min_{\boldsymbol{w}} \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \sum_i \boldsymbol{\alpha^i}^{\mathrm{T}}(\boldsymbol{\psi^i} - \mathbf{1}\boldsymbol{\psi}^{\boldsymbol{i}}_{y^i}) \\
& - \sum_i f^{\star i}(\boldsymbol{\alpha}^i), && (23)
\end{aligned}$$

Table 2: Training times.

$\frac{\lambda}{N}$	WINE		GLASS		VEHICLE		SEGMENT	
	$\sigma^2 = 10.0$ err $= 1.72$		$\sigma^2 = 10.0$ err $= 20.24$		$\sigma^2 = 10.0$ err $= 26.42$		$\sigma^2 = 0.1$ err $= 0.0395$	
	CONJ GRAD.	SMO	CONJ GRAD.	SMO	CONJ GRAD.	SMO	CONJ GRAD.	SMO
0.001	26.47	12.39	38.37	93.03	1951.9	216.69	1169.4	39.33
0.01	13.92	6.50	23.96	7.64	1571.4	26.04	782.7	63.56
0.1	4.66	1.44	1.12	0.93	225.1	4.96	681.4	27.70
1	1.46	0.39	4.02	0.58	227.7	4.66	441.5	28.06
10	1.08	0.38	1.66	0.30	72.6	1.15	259.5	28.41
100	0.70	0.26	1.36	0.30	34.5	1.11	237.5	27.53
1000	0.44	0.24	0.89	0.28	16.5	1.09	186.9	27.90

where $f^{\star i}(\boldsymbol{\alpha^i}) = \sum_y \alpha^i_y \log \alpha^i_y$ is the Fenchel conjugate of $f^i(\boldsymbol{\xi^i})$ (see *e.g.* [2]). Note that the domain of $f^{\star i}(\boldsymbol{\alpha^i})$ is constrained by $\sum_y \alpha^i_y = 1$ and $\alpha^i_y \geq 0$.

We also get the KKT conditions:

$$\frac{\partial L}{\boldsymbol{w}} = 0 \quad \rightarrow \tag{24}$$

$$\boldsymbol{w} = -\frac{1}{\lambda} \sum_{i,y} \alpha^i_y \phi((\boldsymbol{x^i}, y) - \phi(\boldsymbol{x^i}, y^i))$$

$$\frac{\partial L}{\boldsymbol{\xi^i}} = 0 \quad \rightarrow \quad \xi^i_y = \log(\alpha^i_y) - \log \sum_{y'} \exp(\xi^i_{y'}) \tag{25}$$

$$\frac{\partial L}{\boldsymbol{\alpha^i}} = 0 \quad \rightarrow \quad \xi^i_y = \boldsymbol{w}^{\mathrm{T}} \phi(\boldsymbol{x^i}, y) \tag{26}$$

We can now eliminate $\boldsymbol{w}$ by plugging 24 into 23. Finally, stacking all $\boldsymbol{\alpha^i}$ into a vector $\boldsymbol{\alpha}$ and defining the inner product matrix: $K_{(i,y),(j,y')} := \phi(\boldsymbol{x}^i, y)^{\mathrm{T}} \phi(\boldsymbol{x}^j, y')$ yields the dual Program 5.

B Derivatives of the 1-d objective

At each iteration of smo, we solve the one-dimensional problem given in Eq. 9. To solve using Newton-Raphson, we need the first and second derivatives of the objective function. Assuming that $-\alpha^l_{y^1} < \Delta < \alpha^l_{y^2}$ (ie. that the constraints are satisfied strictly), the first derivative is given by

$$Q'(\Delta) = -2a\Delta - \log\left(\frac{\alpha^l_{y^1} + \Delta}{\alpha^l_{y^2} - \Delta}\right) - b, \tag{27}$$

and the second derivative by

$$Q''(\Delta) = -\frac{\alpha^l_{y^1} + \alpha^l_{y^2}}{(\alpha^l_{y^1} + \Delta)(\alpha^l_{y^2} - \Delta)} - 2a. \tag{28}$$

References

[1] Yasemin Altun, Thomas Hofmann, and Alexander J. Smola. Gaussian process classification for segmenting and annotating sequences. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, page 4, New York, NY, USA, 2004. ACM Press.

[2] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, New York, New York, 2004.

[3] Robert G. Cowell, Steffen L. Lauritzen, A. Philip David, and David J. Spiegelhalter. *Probabilistic Networks and Expert Systems.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[4] Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292, 2002.

[5] K. B. Duan. *Improved kernel methods for classification.* PhD thesis, 2003.

[6] T. Friess, N. Cristianini, and C. Campbell. The kernel adatron algorithm: a fast and simple

learning procedure for support vector machine, 1998.

[7] T. Jaakkola and D. Haussler. Probabilistic kernel regression models, 1999.

[8] S. S. Keerthi, K. B. Duan, S. K. Shevade, and A. N. Poo. A fast dual algorithm for kernel logistic regression. *Machine Learning*, 61(1-3):151–165, 2005.

[9] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning*, pages 282–289. Morgan Kaufmann, San Francisco, CA, 2001.

[10] John Lafferty, Xiaojin Zhu, and Yan Liu. Kernel conditional random fields: representation and clique selection. In *ICML '04: Twenty-first international conference on Machine learning*. ACM Press, 2004.

[11] John C. Platt. Using analytic qp and sparseness to speed training of support vector machines. In *Proceedings of the 1998 conference on Advances in neural information processing systems II*, pages 557–563, Cambridge, MA, USA, 1999. MIT Press.

[12] Ben Taskar. *Learning Structured Prediction Models: A Large Margin Approach*. PhD thesis, 2004.

[13] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *ICML '04: Twenty-first international conference on Machine learning*, New York, NY, USA, 2004. ACM Press.

[14] J. Zhu and T. Hastie. Classification of gene microarrays by penalized logistic regression. *Biostatistics*, 5:427–443, 2004.

[15] Ji Zhu and Trevor Hastie. Kernel logistic regression and the import vector machine. *Journal of Computational & Graphical Statistics*, 14(1):185–205, March 2004.